Get Started with Apache Hive

Learning Outcomes

What is Hive?

Apache Hive is a data warehouse framework for querying and analysis of data stored in HDFS. It provides SQL-like declarative language, called HiveQL, to express queries.

It abstracts the complexity of Hadoop MapReduce. Basically, by providing a mechanism to project structure onto the data and perform queries written in HQL (Hive Query Language) that are similar to SQL statements.

Internally, these HQL queries get converted to map reduce jobs by the Hive compiler meaning you don’t need to worry about writing complex MapReduce programs to process your data using Hadoop.

In general, Hive is targeted towards users who are comfortable with SQL. Using Hive-QL, users associated with SQL can perform data analysis very easily.

Hive History

Before 2008, all of the data processing infrastructure at Facebook was built around a data warehouse using RDBMS. The data at Facebook was growing at the rate of 15 TBs/day in 2007, and in a few years, it increased around 2 PB/day.

The infrastructure at that time took a day to process the daily data processing jobs. So they were searching for the infrastructure that scales along with their data.

Due to Hadoop being open-source and scalable, Facebook started using Hadoop. With Hadoop, they were able to process the daily data processing jobs within a few hours.

However, using Hadoop was not easy for end-users, especially for the ones who are not familiar with the MapReduce. End-users had to write complex MapReduce jobs for simple tasks like counts, averages, etc.

So Facebook tried to bring the query capabilities to the Hadoop while still maintaining the extensibility and flexibility of Hadoop. This led the data infrastructure team at Facebook to develop a Hive. Hive then got very popular with all the users internally at Facebook.

Later on, in August 2008, Hive was open-sourced.

Why use Hive?

Apache Hive combines the advantage of both the SQL DataBase System and the Hadoop MapReduce framework. With Hive, one can smoothly perform data analysis without writing the complex MapReduce jobs.

We can use Hive for analyzing, querying, and summarizing large volumes of data. It is best suited for traditional data warehousing tasks where users can perform data analytics and data mining that does not require real-time processing.

It is easily possible to couple Hive queries to different Hadoop Packages like RHive, RHipe, and even Apache Mahout. For example, We can use Tableau and Hive integration for Data Visualization; Apache Tez, along with Hive, will provide real-time processing capabilities, etc.

Hive SQL can be extended with user-defined functions.

Hive Architecture

The major components of Apache Hive are:

- **Hive Client**

  Hive provides support for the applications written in any programming language like C++, Python, Java, etc. by using the JDBC, ODBC, and the Thrift drivers, for performing any queries on the Hive.

- **Hive Services**
  To perform all queries, Hive provides various services like Beeline shell Hive Server2, etc.

  - **Beeline Shell** is a command shell provided by Hive Server 2 that allows users to submit hive queries and commands.

    - **Hive Server 2** enables clients to execute the queries against Apache Hive.

- **Processing and Resource Management**
  Internally, Hive uses the MapReduce framework as the defacto engine in order to execute the queries.
- **Distributed Storage**
  Hive uses the Hadoop Distributed File System for distributed storage.

Hive Features

1. Apache Hive provides easy access to data through SQL like queries, thus we can use it for data warehousing tasks such as ETL (extract/transform/load), reporting, and data analysis.
2. Using Apache Hive, we can impose structure on a variety of data formats.
3. Apache Hive facilitates access to the files stored either directly in HDFS or in other data storage systems such as HBase.
4. Query execution via Apache Spark, Apache Tez or MapReduce.

Limitations of Hive

1. Hive is best for working with batch jobs, so we cannot use it for Online Transaction Processing. We can use it for Online Analytical Processing.
2. Hive does not offer real time queries for row level updates.
3. There can be a delay while performing Hive queries.

Practice Exercises

Exercise 1

Run each of the following commands on the shell and describe the outcomes of each one of them.

1. `hive`

2. `SHOW databases;`
3. `SHOW tables;`
4. `CREATE DATABASE test_hive;`
5. `SHOW databases;`
6. `DROP DATABASE test_hive;`
7. `SHOW databases;`
8. `!quit`

Exercise 2

Use uploaded data to build a table in Hive using the following commands.

Pre-requisites

1. Login into Apache Ambari and then visit "Files View" by clicking the tiles icon on the top right of the navigation bar.
2. Create a new directory `'/user/working-files'` and upload the following CSV files (**ratings**: https://bit.ly/hiveratings and **movies**: https://bit.ly/hivemovies, into that directory.
   - We'll import this data into Hive and practice running HiveQL.
3. SSH into your Hadoop instance and run the following queries.

Tasks

1. `hive`
2. `CREATE TABLE movies (id INT, title STRING, genres STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' stored as textfile;`
3. `LOAD DATA INPATH '/user/working-files/movies.csv' OVERWRITE INTO TABLE movies;`
4. Repeat the above queries for `ratings.csv`.

Explanation

- **ROW FORMAT DELIMITED** means that each line is a row inside the text file.
- **FIELDS TERMINATED BY ','** means that the columns are separated by a comma.

- **STORED AS TEXTFILE** means that the data is stored in text files.
- **LOCATION** means that it is located in HDFS at a the specified location.

Exercise 3

Run the following queries to perform data analysis. Describe the outcomes of each query.

1. `hive`
2. `DESCRIBE movies;`
3. `DESCRIBE ratings;`
4. `SELECT * FROM movies LIMIT 10;`
5. `SELECT * FROM ratings WHERE userId = 149;`
6. `SELECT movieId,rating FROM ratings WHERE userId=149;`
7. `select movieId,rating,name from ratings join movies on ratings.movieId=movies.Id where userId=149;`
8. `SELECT AVG(rating) FROM ratings WHERE userId=149;`
9. `SELECT userId, COUNT(userId), AVG(rating) FROM ratings GROUP BY userId;`
10. `CREATE TABLE USERRATING (userId INT, numratings INT, avgrating FLOAT);`
11. `insert overwrite table userrating SELECT userId, COUNT(userId), AVG(rating) FROM ratings GROUP BY userId;`